Feasibility exploration of partial Reconfigurable FPGA for REAL products

Robbie Vincke
Nico De Witte
Jeroen Boydens

Report CW 653, December 2013

KU Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Feasibility exploration of partial Reconfigurable FPGA for REAL products

Robbie Vincke
Nico De Witte
Jeroen Boydens

Report CW 653, December 2013

Department of Computer Science, KU Leuven

Abstract

Partial Reconfiguration is the ability to dynamically modify blocks of logic by downloading partial bit files while the remaining logic continues to operate without interruption. The concept is analogue to a processor context switch.
- System Flexibility: When a specific part of a design needs to be reconfigured it is sometimes necessary to preserve the existing communication link instead of resetting the full device.
- Size and Cost Reduction: Some function are time-mutual exclusive to each other. This means some functions never need to exists on the same time. Instead of implementing all functions in parallel and selecting the needed function using a multiplexer, PR can dynamically change the needed function.
- Power Reduction: In embedded systems where power efficiency is an issue. Some functions can be reconfigured with a blank bitstream to save power consumption. Also multiple versions of the same function can be made. A high-end implementation consuming a lot of power and a minimal implementation consuming much less power.

Keywords : dynamic partial reconfiguration, programmable logic, reconfigurable design.
CR Subject Classification : B.3.6

KU Leuven - Kulab

FoRReal

Feasibility exploration of partial Reconfigurable FPGA for REAL products

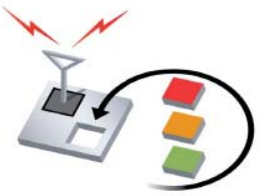
Contents

List of Figures

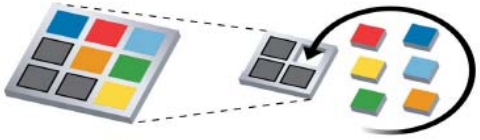
Introduction

Partial Reconfiguration is the ability to dynamically modify blocks of logic by downloading partial bit files while the remaining logic continues to operate without interruption. The concept is analogue to a processor context switch.

- System Flexibility: When a specific part of a design needs to be reconfigured it is sometimes necessary to preserve the existing communication link instead of resetting the full device.



- Size and Cost Reduction: Some function are time-mutual exclusive to each other. This means some functions never need to exists on the same time. Instead of implementing all functions in parallel and selecting the needed function using a multiplexer, PR can dynamically change the needed function.



- Power Reduction: In embedded systems where power efficiency is an issue. Some functions can be reconfigured with a blank bitstream to save power consumption. Also multiple versions of the same function can be made. A high-end implementation consuming a lot of power and a minimal implementation consuming much less power.

Example use-cases

The next paragraphs provide an overview of some example use cases where partial reconfiguration can come in handy. (1) Communication Hub (2) Network Multiport Interface and (3)

Communication Hub

As mentioned in the system flexibility part above. PR makes it possible to preserve communication links while reconfiguring other parts of the FPGA. The system illustrated below has three main communication links: (1) A radio link, (2) a video link with a monitor connected to it and (3) a bus link, for example PCI Express. The application logic itself consists of three PR regions. Each PR region can be reprogrammed with a partial bitstream. The number of combinations which can be constructed are

$N_{PR1} * N_{PR2} * N_{PR3}$ where N_{PRX} is the number of available partial bitstreams for a PR region X.

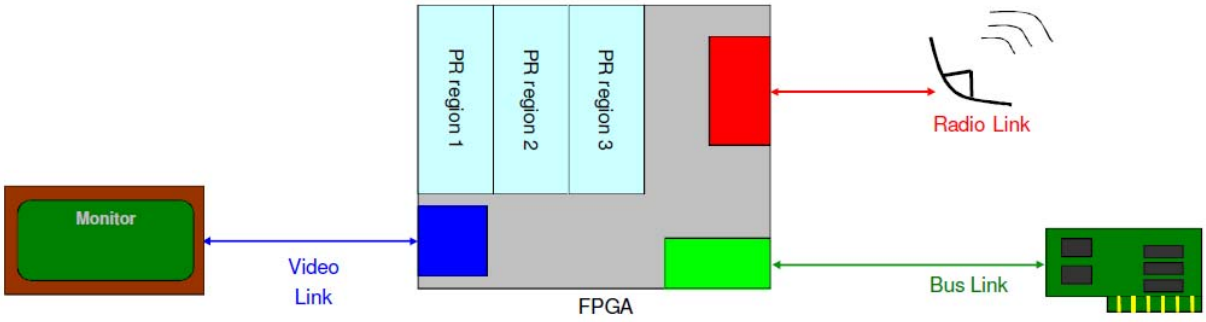


Figure 1. Communication Hub

Networked Multiport Interface

To illustrate the size- and cost reduction of PR designs a telecommunication system is implemented with 4 ports, each port can implement a communication protocol (10 GigE, OC48, Fibre or a custom one). It is however unknown which port needs which protocol. And it can happen that more than one port must implement the same protocol. Instead of implementing all protocols in parallel for each port, one can use PR. External memory is used to store all partial bitstreams. Each port can be configured on the fly with the requested bitstream as illustrated in Figure 2. Networked Multiport Interface illustrating size and cost reduction of PR designs

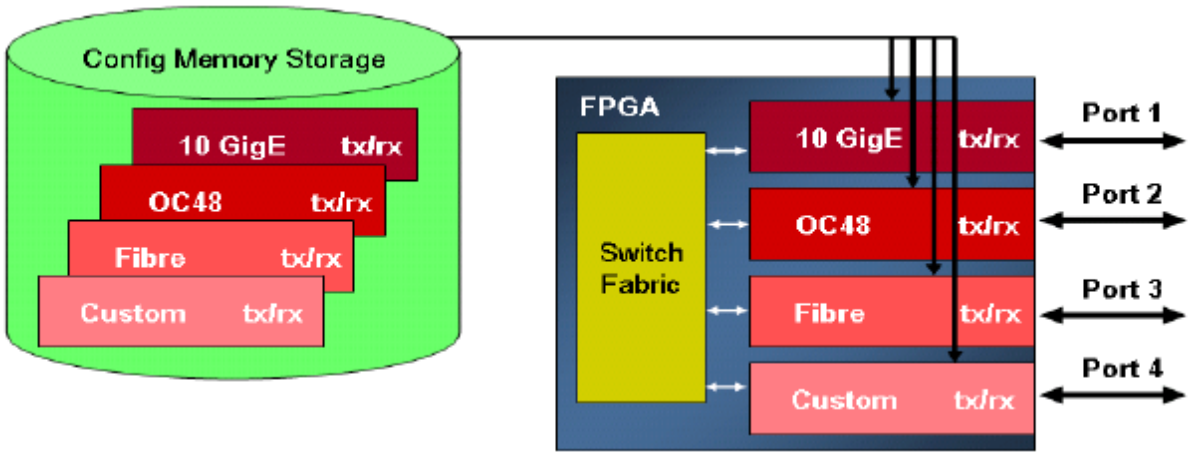


Figure 2. Networked Multiport Interface illustrating size and cost reduction of PR designs

Configuration via PCI Express Interface

For larger devices it is difficult to meet the PCI Express enumeration time of 100ms[1]. The initial configuration time can be reduced using two-stage configuration. The initial configuration is done using a compressed bitstream in external memory. When the requirements are met, as second stage, the partial bitstream containing the user application can be configured through the PCI Express interface.

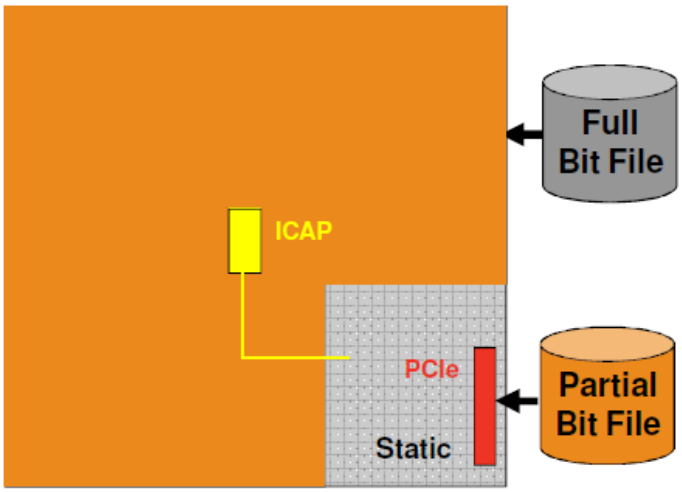


Figure 3. PCI Express configuration

This type of configuration is often referred to as partial configuration since its only goal is to bring up the full system in multiple stages. Once the system is completely configured it is not reconfigured again.

Terminology

- Partition: A logical block (entity or instance) to be used for design reuse.
- Bottom-up synthesis: Separate synthesis projects into multiple netlists. This means there is no optimization across projects.
- Reconfigurable Partition: Design hierarchy instance marked by user for reconfiguration. EDA Tools map all logic within the boundaries of these partitions. Altera Tools need the partition to be marked as LogicLock region for the same reason.
- Reconfigurable Module: A specific logical design that occupies a reconfigurable partition. A reconfigurable module is equivalent to Altera's "persona".
- Static Logic: All logic in the design that is not reconfigurable.
- Configuration: Consist of static logic and a certain reconfigurable module for each reconfigurable partition.
- Partition Pins: Pins of the interface between static en reconfigurable logic.
- Proxy Logic: A LUT inserted on each partition pin to act as anchor point for reconfigurable partitions.

Configuration Types

- "Standard" Configuration: When Vcc is stable the system goes into power-on reset state. Next a complete configuration bitstream is loaded from, for example, external memory. When the FPGA is configured it goes into user mode for normal operation.

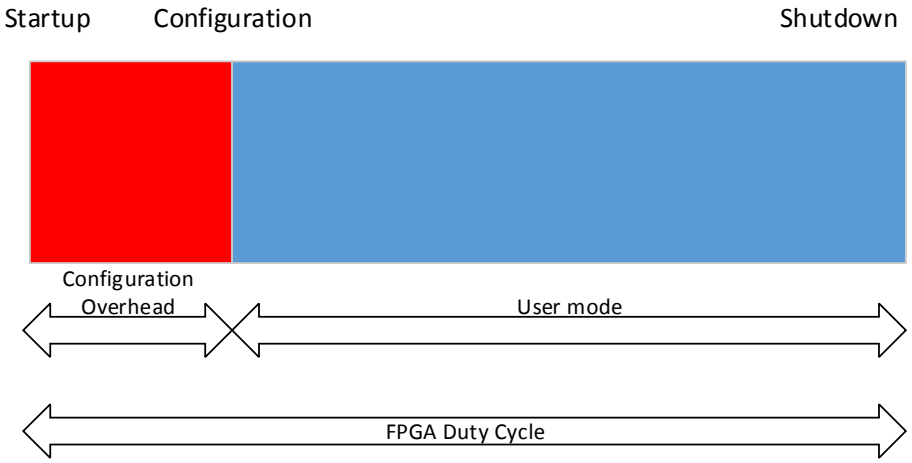


Figure 4: Standard Configuration

- Full reconfiguration: When Vcc is stable at startup the initial configuration is done. Next the FPGA goes into user mode. When needed a full reconfiguration is done affecting the complete FPGA. When the reconfiguration is done the FPGA goes back to user mode.

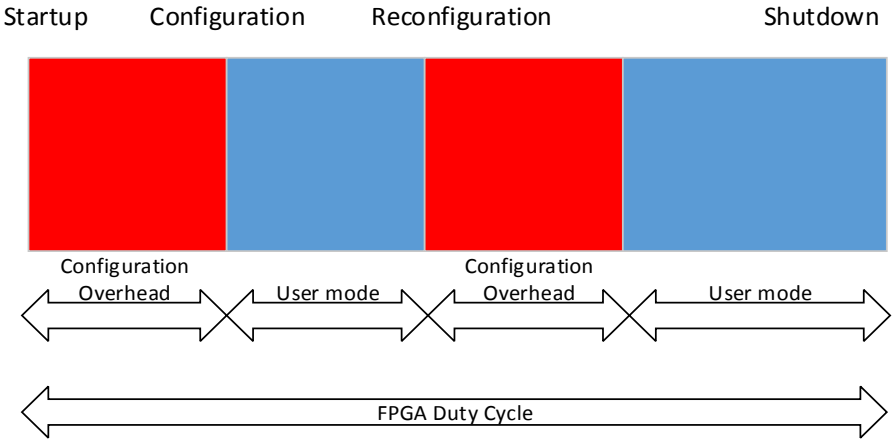


Figure 5: Full reconfiguration

- Static Partial Reconfiguration: As in the other cases when Vcc is stable an initial bitstream is loaded. Next the FPGA goes into user mode. When needed a part of the FPGA is reconfigured with a partial bitstream. During the partial reconfiguration the unaffected parts of the FPGA go into reset state.

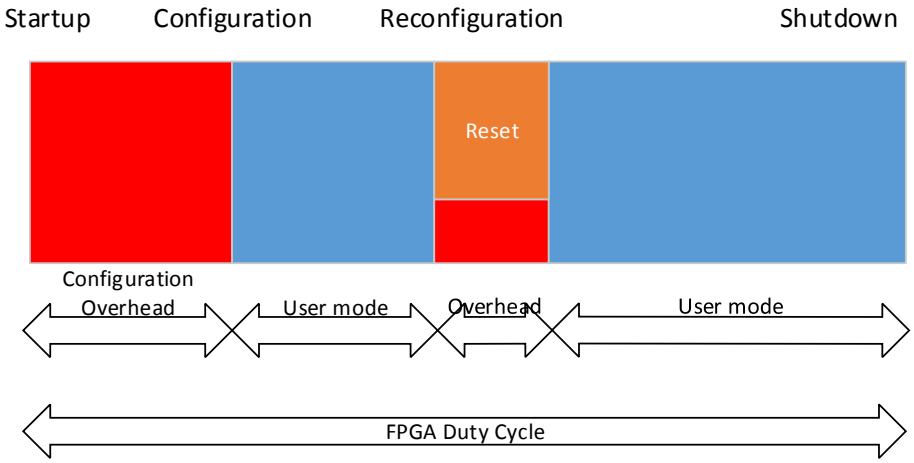


Figure 6: Static Partial Reconfiguration

- Dynamic Partial Reconfiguration: When Vcc is stable an initial bitstream is loaded. Next the FPGA goes into user mode. When needed a part of the FPGA is reconfigured with a partial bitstream. During the partial reconfiguration the unaffected parts of the FPGA continue to function as normal.

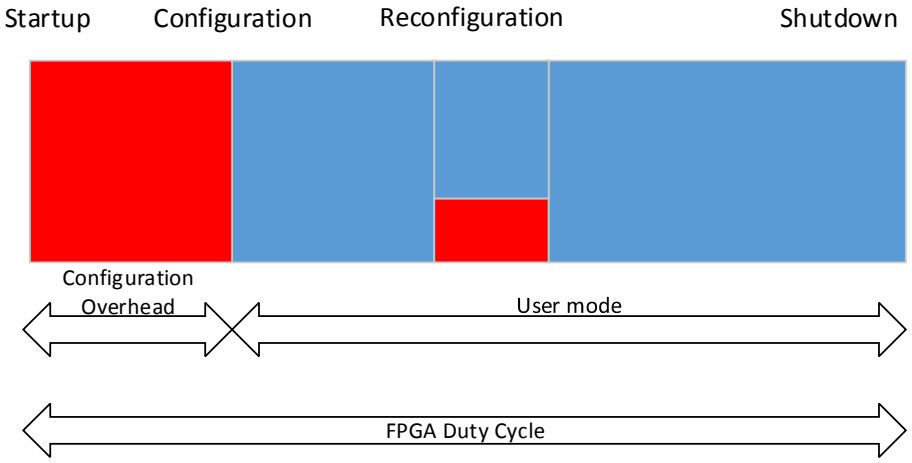


Figure 7: Dynamic Partial Reconfiguration

- Partial Configuration: When Vcc is stable an initial bitstream is loaded only containing a part of the functionality. Next, the FPGA goes into user mode. The partial bitstream is configured immediately after the initial configuration to add functionality to the system. It is a multi-stage bring up of the system.

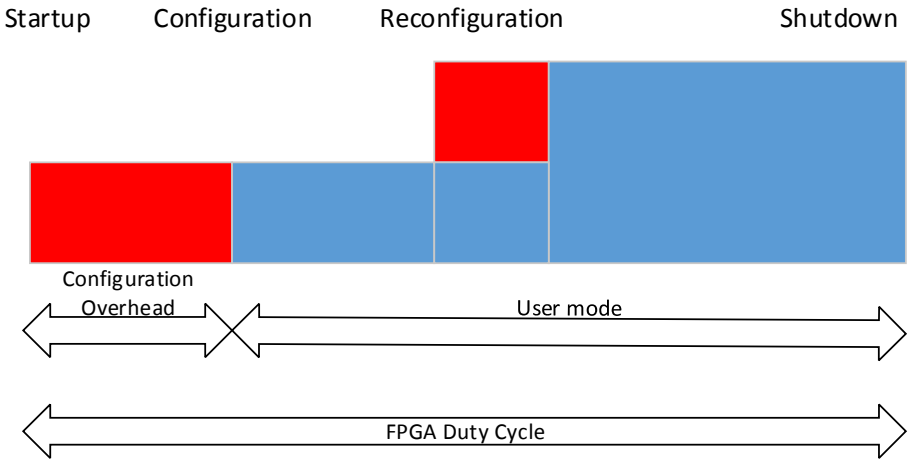


Figure 8: Partial Reconfiguration

Limitations

Since defining partitions restricts the automated routing mechanism the maximum reachable frequency is about 80% of the original maximum frequency. A rule of thumb is to use only 80% of the available logic to give some freedom to the automated place-and-routing tool.

Current state-of-technology

Altera

Altera should support Partial Reconfiguration on all their devices ranging from the low-end Cyclone FPGA's to the high-end Stratix devices. However on the moment of writing only Stratix V devices support the partial reconfiguration feature. The online available information is rather limited.

Xilinx

Xilinx is relatively farfetched with the incorporation of the feature in their design tools. Especially The toolflow based on MicroBlaze system-on-chips and software initiated reconfiguration using the ICAP interface is very well documented. It is supported on all 7-series devices. In earlier generations only the Virtex devices supported partial reconfiguration. However there are academic tools available for reconfiguring Spartan 6 devices. Examples are GoAhead from the university of Oslo[2]. On the moment of writing PlanAhead is the driving tool after partial reconfiguration. The Vivado tool suite does not support it yet.

Design considerations

Primitives

Not everything in a FPGA is reconfigurable. The reconfigurable modules can only consist of these primitives:

- Slice logic (LUTs, flip-flops)

- Memories (block RAM, distributed RAM, shift register LUTs)
- DSP blocks

Logic that must remain in static logic are:

- Clock-modifying blocks (MMCM, DCM, PLL, PMCD)
- Global clock buffers (BUFG)
- I/O components (IOLOGIC, IODELAY, IDELAYCTRL)
- Device feature blocks (BSCAN, ICAP, STARTUP, or PCIE, for example)

The reason is that a partial reconfigurable primitive must be controlled by configuration memory. I/O components are controlled by I/O configuration bits.

It is however, especially for clock-modifying blocks, possible to use the dynamic reconfiguration port (DRP) (note the difference between dynamic reconfiguration and dynamic partial reconfiguration). More information on the DRP can be found in [3] for Xilinx and in [4] for Altera. Further details on DRP is beyond the scope of this document.

Physical Location

Loading a partial bitstream does not require knowledge of the physical location of the reconfigurable module. The frame addressing is embedded in the partial bitstream itself.

Building a PR Design using the Xilinx Tools

Toolflow example

The toolflow is presented using an audio filter application implemented in a Xilinx Virtex 5 FPGA. The audio samples of the source are filtered using Matlab generated filter implementations (low-pass, high-pass and all-pass). This is done for both audio channels.

The reconfiguration is user controlled using the UART peripheral en a MicroBlaze soft-processor. Partial bitstream are located on a Compact Flash card.

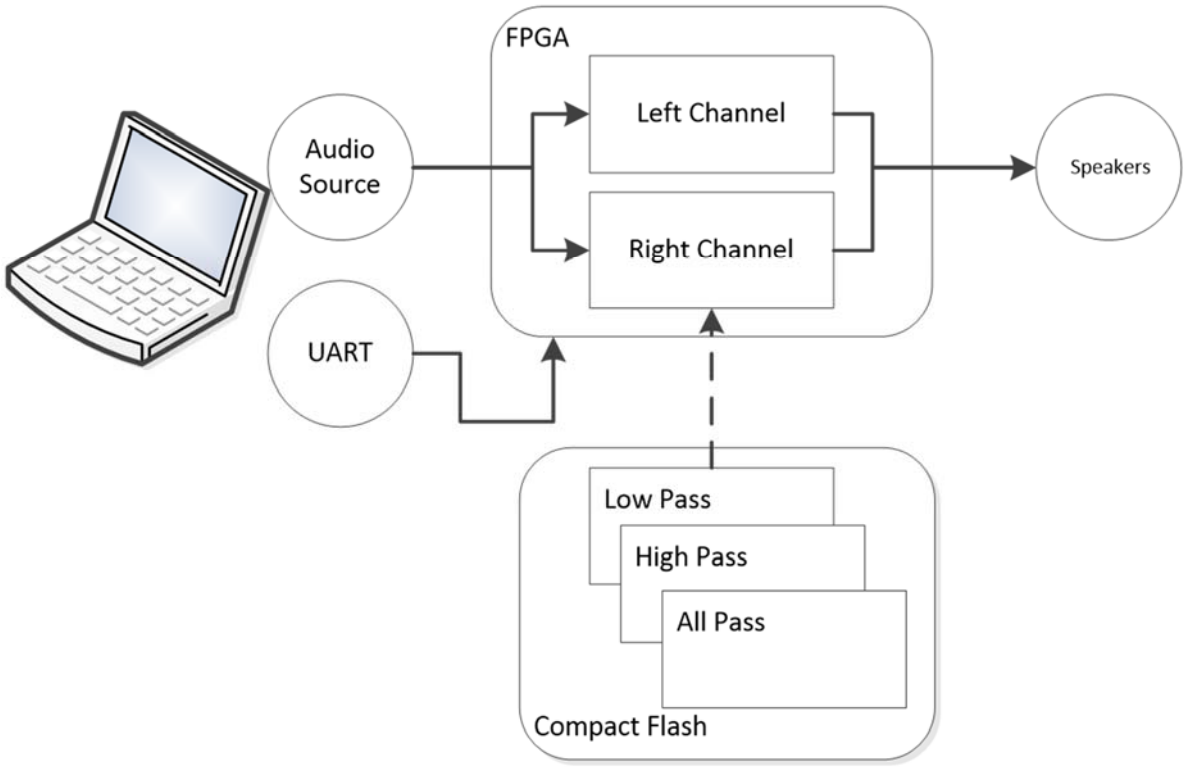


Figure 9: Audio filter reconfigurable system

1. Netlist generation
   To implement partial reconfiguration the netlists of all used peripherals should be available therefore each configuration of the reconfigurable module is synthesized on beforehand. The audio filters are pre-synthesized using Matlab SysGen plugin. Other peripherals are imported from the pre-existing IP core library offered by Xilinx. These peripherals are: (1) HWICAP: The configuration access port, (2) SysAce: CompactFlash card controller, (3) AC97: audio codec and (4) UART: serial communication and user interaction.

2. Hardware System
   The hardware system is build using the Xilinx XPS tool. A MicroBlaze soft processor is instantiated and communicates with the peripherals using the Processor Local Bus (PLB).

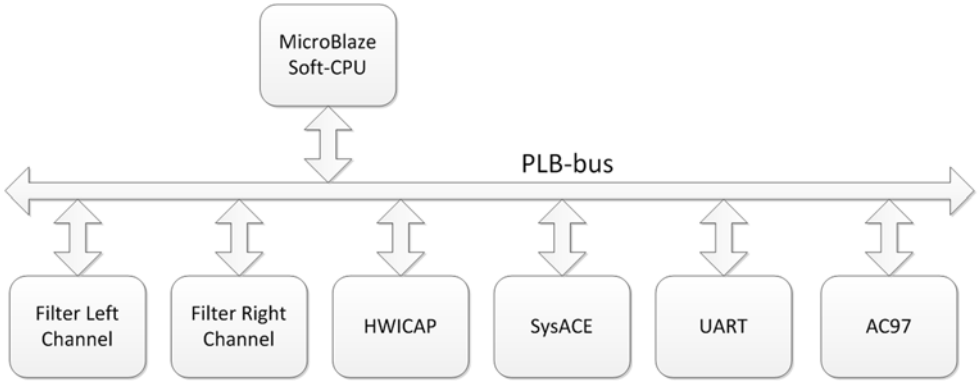


Figure 10: PLB-based SoC system

3. Software
   The software is written in C using the Xilinx SDK tool. The software is responsible for (1) driving the serial communication and user interaction, (2) the interface with AC97 codec and (3) controlling the HWICAP peripheral. Traditional software testing techniques can be used here.

4. Define Partitions
   Select which peripheral is marked as reconfigurable partition. In this case the filter peripheral on both channels is reconfigurable.

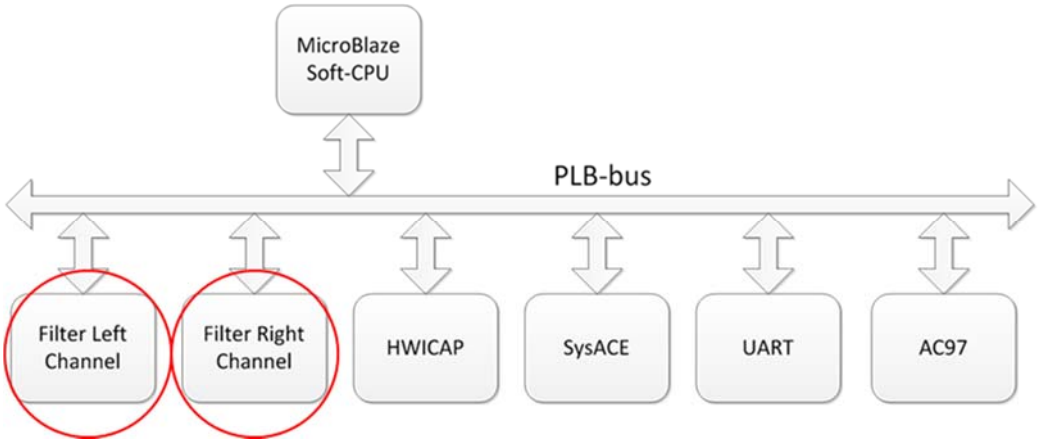


Figure 11: PLB based SoC System with reconfigurable peripherals

5. Modules
   Add different modules to defined partitions by using the pre-synthesized netlists or a blank netlist. In this case 2 netlists are used: (1) lowpass filter netlist and (2) highpass filter netlist.

6. Floorplanning
   The defined partitions should be floorplanned in such manner all required

primitives are available. The dimensions of the partitions should be based upon the most requiring (in terms of resource utilization) module.

7. Design Rule Check
   Xilinx provides a set of design rules to check the floorplanned partitions. Some of the parameters which are checked are: minimum range of a partition, no regions inside the same frame, clocking problems, etc.

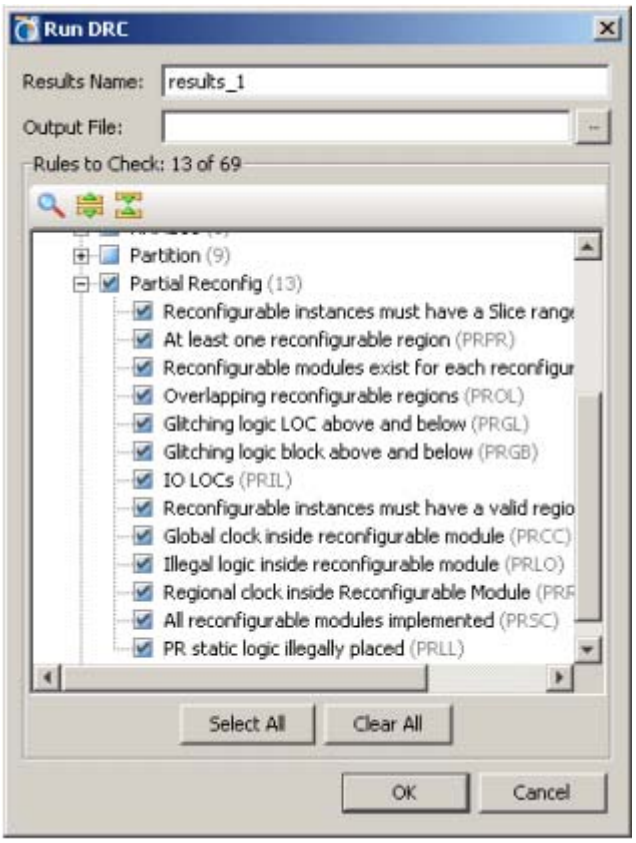


Figure 12: Xilinx Design Rule Checks

8. First configuration
   Use the most requiring module to implement the first configuration. According to Xilinx this is necessary to prevent routing and timing problems in the future. The main philosophy is that if it works for the most requiring module it will work for less demanding modules too.

9. Promote configuration
   After implementation the result should be “promoted”. This means the static logic remains preserved for other configurations.

10. Other configurations
    Implement all configurations with different module combinations.

11. PR_Verify
    Another Xilinx provided tool is “PR_Verify”. This tools checks all routing and placement of the static logic and its consistency among different configurations. All module interfaces (proxy logic) should be the same for all different configurations.

12. Bitfiles
    For each configurations bitstreams can be generated. This include full bitstreams and partial bitstreams per configuration.

13. Generate CompactFlash image
    A CompactFlash card image should be generated to store partial bitstreams and software application.

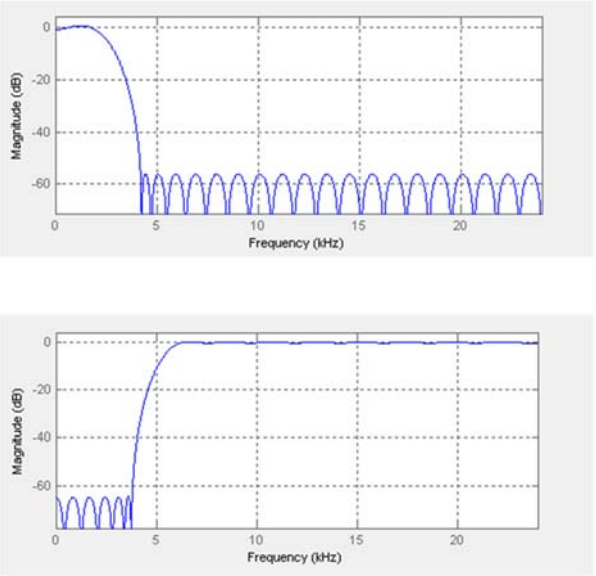


Figure 13: Audio Filter design

General flow for Non-CPU based systems

First you need a top-level HDL design instantiating all needed components (static and reconfigurable modules). It is desirable to name all reconfigurable components with the “rModule” prefix for readability. All rModules should have exactly the same interface. The toplevel project only contains a toplevel HDL file instantiating unknown components (all sub-components should be marked with a questionmark in Project Navigator). The Synthesis properties should be configured to add I/O Buffers tot the design since it is the top-level design. Next, the toplevel design should be synthesized resulting in a netlist-file (NGC).

Next all static logic HDL files should be grouped in a new Project Navigator project. In turn, each HDL file should be marked as top-level HDL and synthesized (keep in mind that the addition of I/O buffers should be disabled now) separately. This results in a number of netlist files equal to the number of static components. The static part

contains control logic. In order to know the exact rModule loaded on a certain time a specific ID output-pin can be used to inform the control state-machine.

Next all rModules should be loaded in separate Project Navigator project. Each should be synthesized separately resulting in one netlist file for each rModule.

Now all design sources are synthesized and ready to be used in a post-synthesis PlanAhead project. Open a post-synthesis project with partial reconfiguration enabled. Add netlist sources from the static design. This mean netlist files of previously synthesized projects (toplevel and static logic) and EDIF/EDN netlists of synthesized IP cores. Add the toplevel user constraints file (UCF) containing pin placement.

When the project is created all netlists should be loaded by opening the synthesized design. Warning messages appear of missing instances which are converted to black boxes. Since we didn't added the rModule netlists yet this is normal behavior. Other netlists might be missing too (VCC, GND netlist from an IP core instance).

The next step is to define reconfigurable partitions. Therefore the missing instances of rModules should be marked as a reconfigurable partition. At first we create the reconfigurable partition without a netlist which creates a black-box module. Next other reconfigurable modules should be added containing the appropriate netlist. No constraint file is needed at first.

Next all reconfigurable partitions should be defined by selecting the region on the physical device. Make sure to over-rate the required LUT's as mentioned by the synthesis report (rule of thumb is add 20% of required LUT's). Make sure no multiple reconfigurable partitions are defined in the same frame. Running Design Rule Checks can check such misplacements.

Next a first design run is created using the most demanding modules (number of LUT's). When the first design run is completed (Place and Route) the implementation can be opened and the placement is visible in the floor plan. If needed additional constraints can be added (delay between static logic and partition). If so, the design run should be reset and run all over again. When the design run is finished the implementation should be promoted. This means the routing and placement of all static logic will be copied for future design runs with other rModules.

Next the other design runs should be created. Add new runs and assign for each run which rModule should be used.

General flow for CPU-based system

The experiments are conducted using a Xilinx MicroBlaze soft-CPU platform. However the design flow is similar for hard-CPU based systems such as Zynq. Start a new Platform Studio project selecting the desired peripherals. Add IP cores using the IP catalog. One of the IP cores is the reconfigurable peripheral. For partial reconfiguration the FPGA Internal Configuration Access Port (ICAP) is needed. Configure the ICAP to instantiate is Startup primitive in the HWICAP core. The primitive is used to generate an End of Startup (EOS) signal and regulate user

access to the ICAP [5]. Next connect the clock generator output port to the clock port of the ICAP device. A storage peripheral is needed for storing the partial bitfiles. SystemAce can be selected for usage of CompactFlash Card. Linear Flash memory can be selected for using flash memory. When all IP cores are selected and connected through the AXI bus netlist files (NGC) can be generated.

Next export the generated hardware to a Software (Xilinx SDK) project. Generate a new Board Support Package (BSP) (select additional software drivers if needed) and add an application project using the generated BSP. A reference software design is available for driving the ICAP peripheral through software.

Start a new PlanAhead project and import generated netlist files and user constraint files. The reconfigurable peripheral is unidentified since no netlist has been associated with the module yet. Select the peripheral and create a reconfigurable partition associated with a black-box bitstream. Next add reconfigurable modules containing the other partial netlists. The next step is to floorplan the partition using the Physical Constraints window. Be sure to add about 20% overhead to the calculated needed resources for routing purposes. After floorplanning run the design rule checker to detect any design rule violations.

Start by implementing the first design run using a custom implementation strategy (-bm c:\path\to\system.bmm). Implement the first configuration and promote the design. Next implement other configurations. After all design runs are implemented be sure to run the PR_verify tool to verify all existing configurations. Finally all bitstreams can be generated.

Industrial Case Study

ZEDBoard Overlay Banner

General

The overall architecture of the application can be described as a reconfigurable video generator. An internal video generator is used to generate test pattern. The generated video is forwarded through a banner generator IP core (which is reconfigurable). The banner generator puts a banner in overlay to the video source. The output of the banner is send to HDMI and VGA.

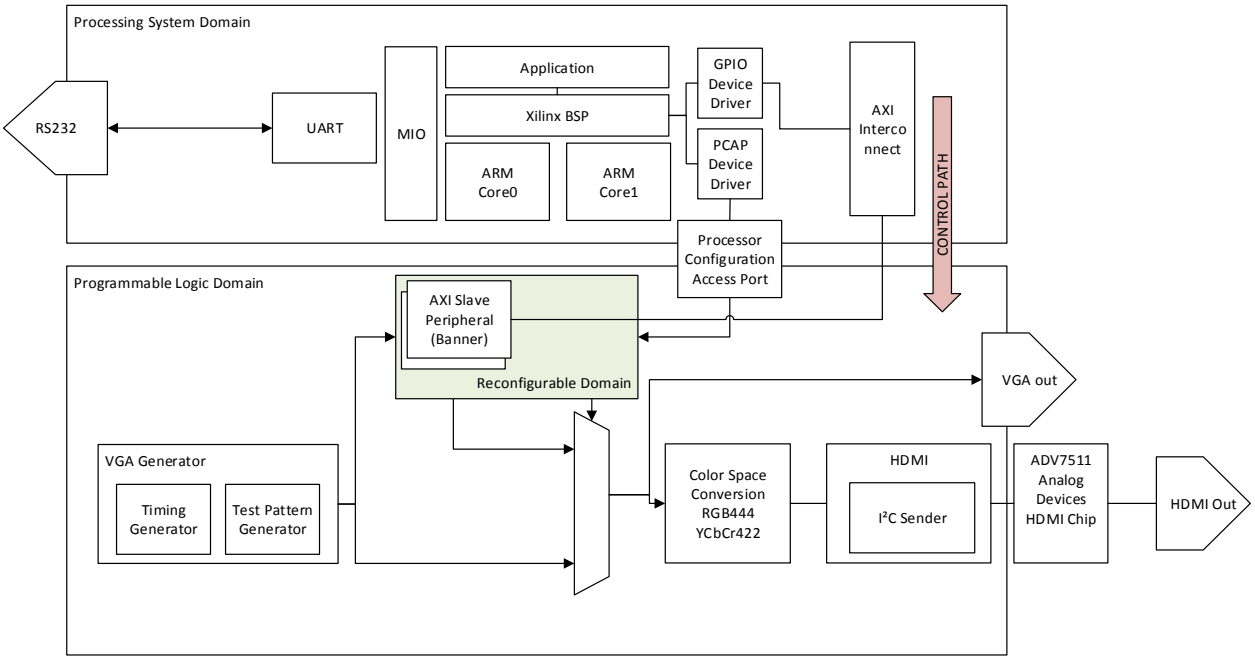


Figure 14: Overlay banner system

Platform

The system is implemented on a Xilinx ZEDBoard featuring a ZC7020 Xilinx Zynq SoC. The video generation and forwarding is done completely in hardware logic. The ARM Cortex-A9 dual-core processor is used to interface with the user over UART and control the reconfiguration access port using a software driver.

Hardware Data Path

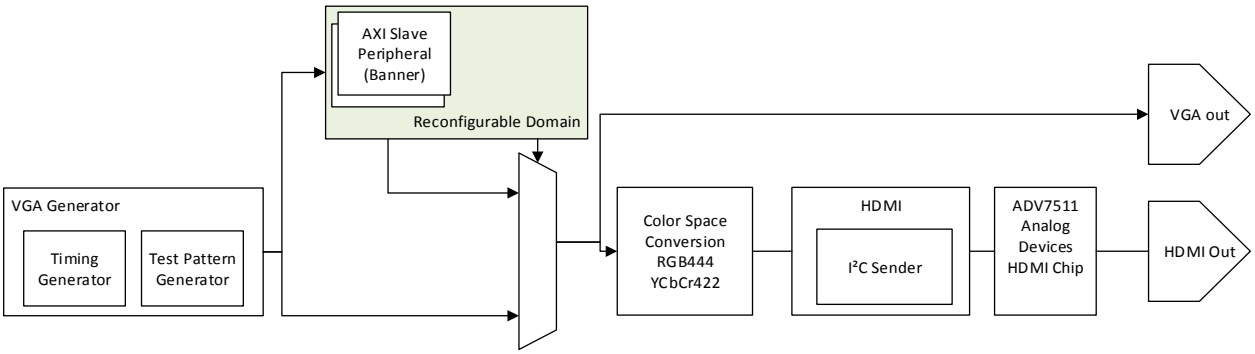


Figure 15: Data Path

The timing generator is used to generate the correct video timings for 720p resolution with a refresh rate of 60 Hz. The test pattern generator generates colored bars. The generated video signal is transferred to the reconfigurable pattern generator.

The pattern generator has as input the horizontal counter and vertical counter as generated by the VGA timing generator. Depending on the values of these counters the RGB values of the pixels, as generated by the test pattern generator, are overwritten with banner information. This is for example a fixed color with a moving square. All RGB values generated by the test pattern generator go through the pattern generator. The resulting RGB values after pattern generator are driven through a multiplexer. This allows us to enable or disable the overlay banner. When reconfiguring the banner generator the datapath can be decoupled from reconfigurable partition in order to preserve video output on the display.

The output of the multiplexer is used for the VGA output and HDMI output. Driving the VGA output is straightforward as the RGB values and sync signals are one-on-one mapped to the VGA pin out. Remark the ZEDBoard only supports 4 bit/color VGA. Therefore the MSB are used to generate VGA colors.

The HDMI path is a bit more complex. The generated video is converted to another color space needed by the ADV7511 HDMI Chip. A conversion from RGB:4:4:4 is done to YCrCb:4:2:2. Next, the I2C controller is used to communicate with the ADV7511 Chip, which is driving the HDMI signals.

Software Control Path

The system can be controlled by software executing on the dual-core ARM Cortex-A9. No operating system is needed to guarantee the fastest reconfiguration times. A bare metal software driver is needed to control the processor configuration access port (PCAP). The software is fairly simple since it reacts on UART command inputs to reconfigure the overlay banner.

As additional feature hardware registers can be made available in the software control path for example to configure the banner color.

Reconfiguration of the design

A first attempt was made by creating a custom IP peripheral containing the full hardware datapath . One toplevel is used for AXI interface communication between the programmable logic and processing subsystem as it is not necessary to configure other entities inside the data path from the processor.

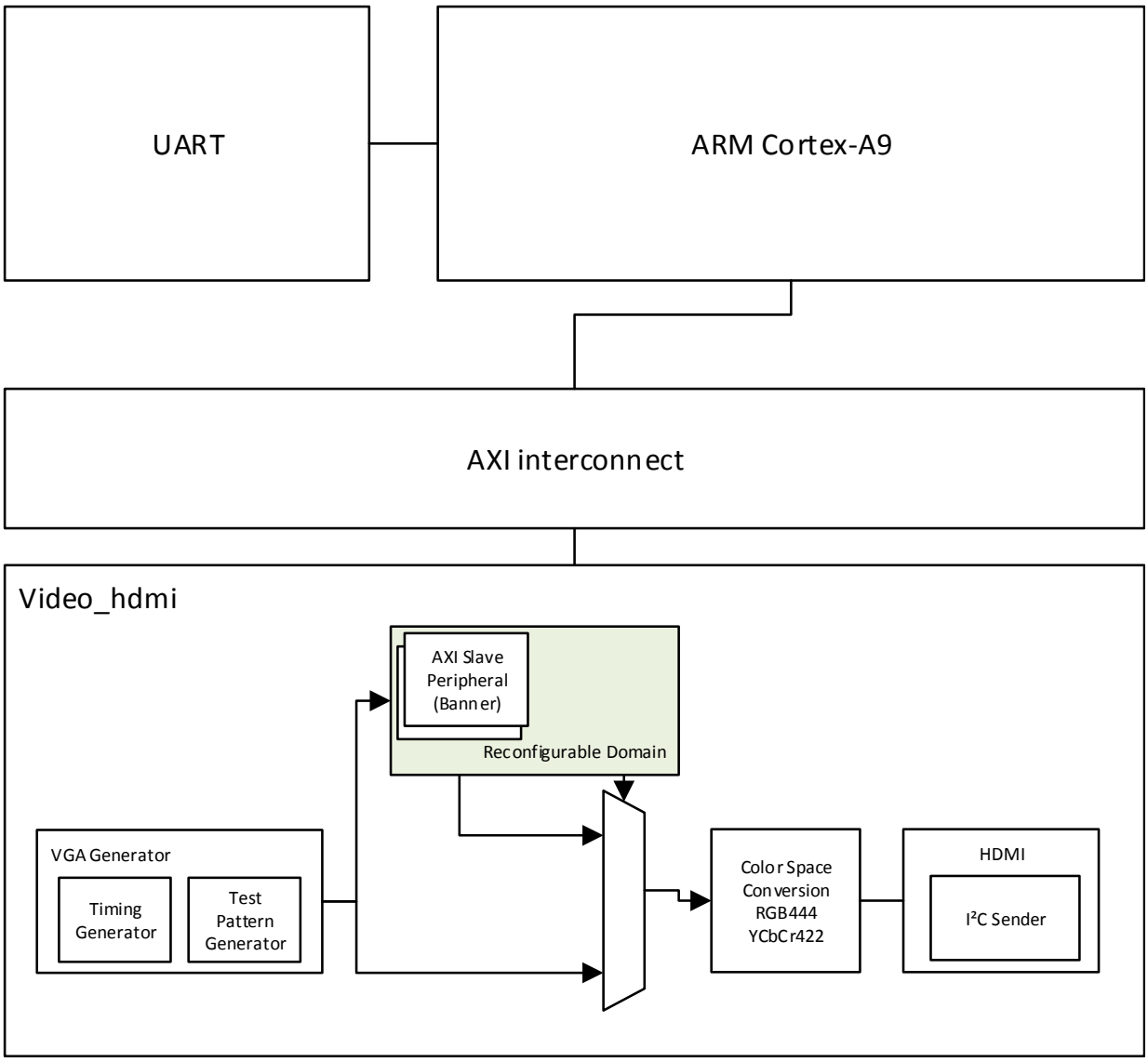


Figure 16: Hardware datapath as one peripheral

The consequence of this design choice is that only one netlist is generated for the complete hardware datapath. Since only one entity inside the hardware data path is reconfigurable separate netlists are needed for that peripheral.

Possible solution are:

1. Extract the reconfigurable module (banner overlay) from the full datapath and connect it as separate peripheral to the system. This would force the tools to generate a netlist for the banner overlay generation seperatly. This is needed for the post-synthesis planAhead project.

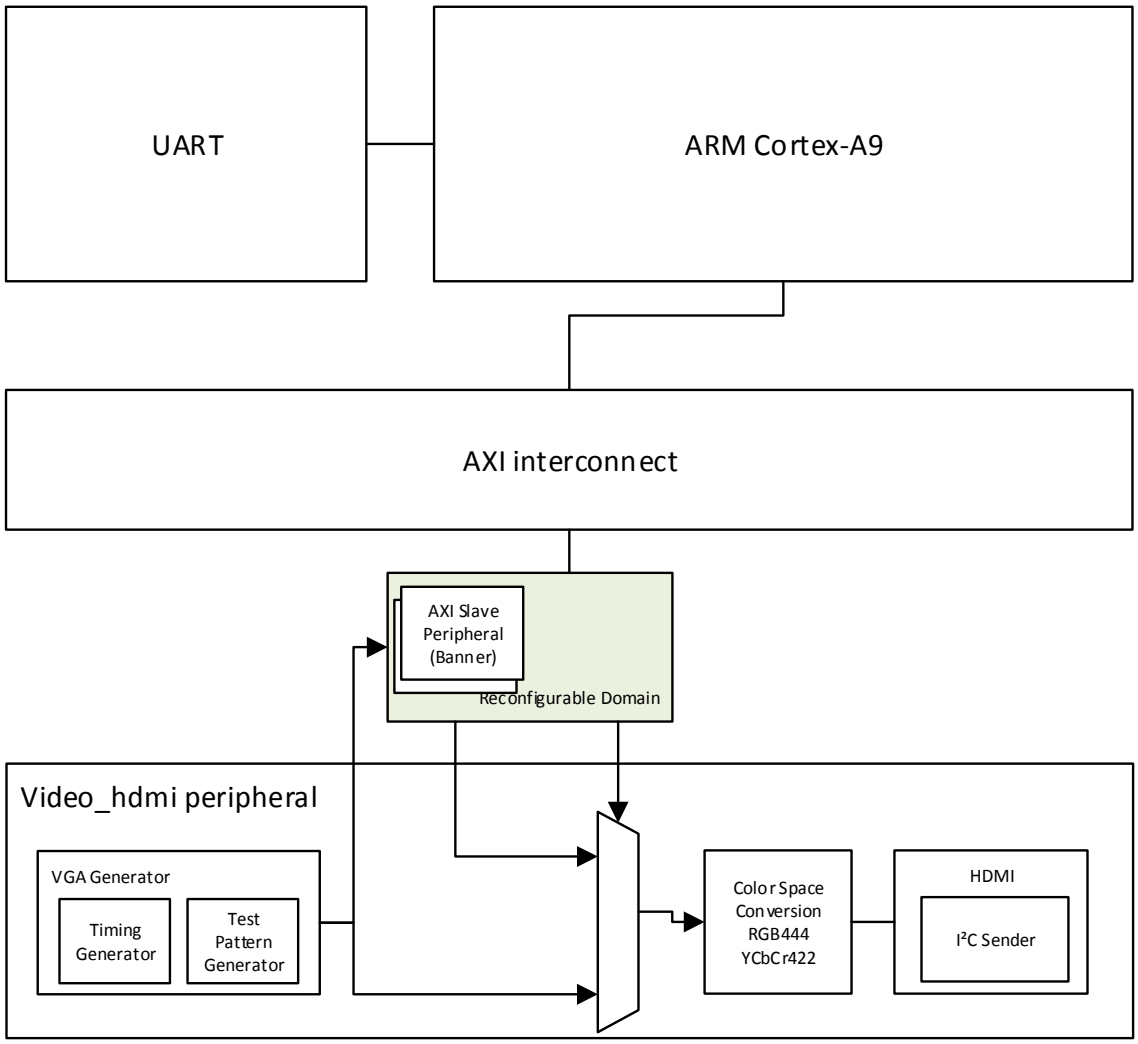


Figure 17: Banner overlay generator as seperate peripheral

2. Attach each entity from the hardware data path as separate peripheral to the AXI interconnect. This mimics the design flow of all Xilinx provided examples. But adds significant design overhead for interfacing each peripheral to the AXI bus.

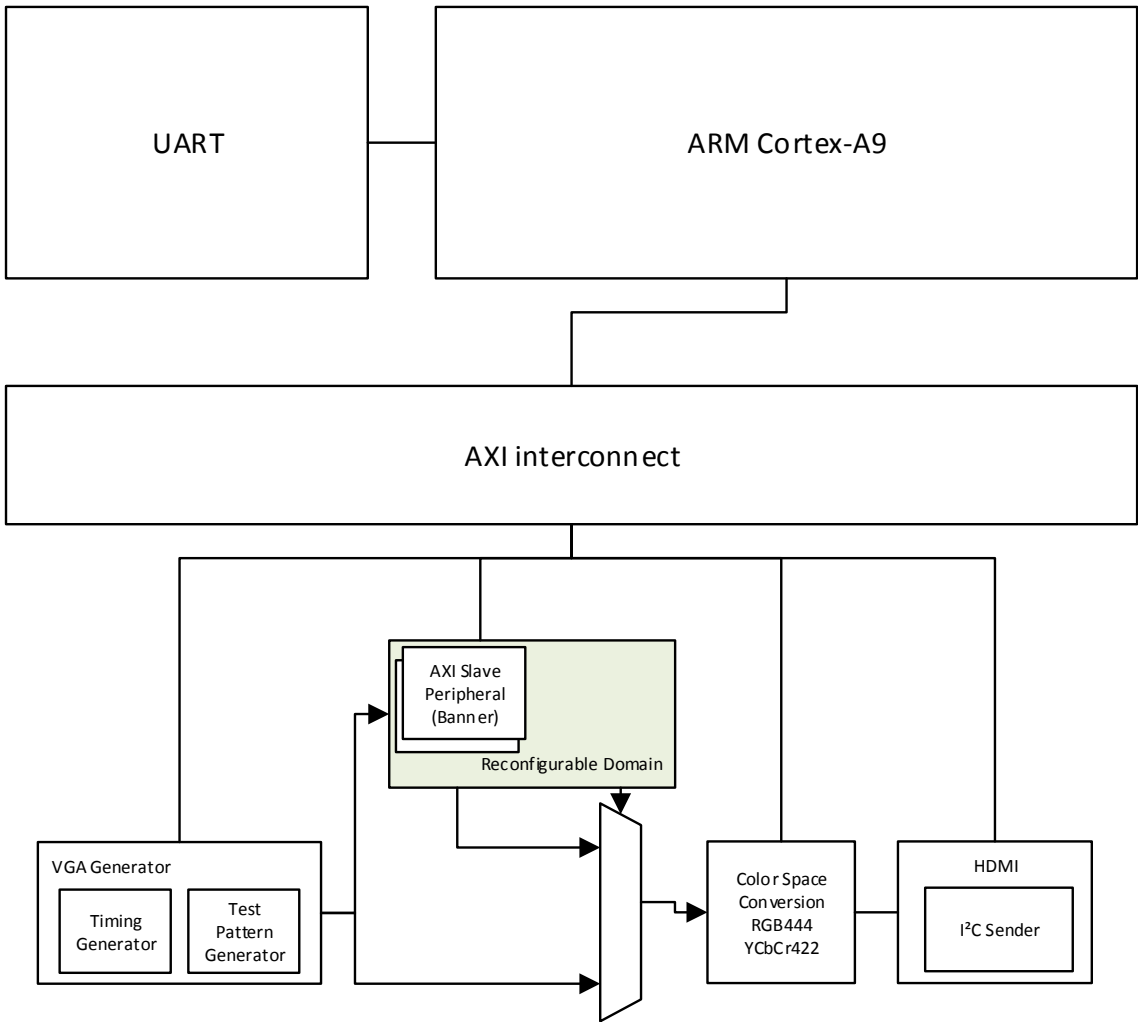


Figure 18: Each data path entity as seperate peripheral

3. The datapath peripherals was developed as Project navigator project. By setting each entity as top-level module separate netlists can be created. It is important to disable the automatic generation of IO Buffers. These netlists could be imported in the post-synthesis Plan Ahead project instead of the general netlist of the peripheral.

   This option is mainly a matter of getting the tools configured correctly. Which can be very challenging in the current toolflow.

Zynq-7000 reconfigurable video filters

General

The purpose of this experiment is to implement reconfigurable high-definition video filters on a real-time input video stream[6]. The input video stream is provided using

a FMC expansion card with a HDMI input with ADV7611 Chip. The output of the video stream is done using the ZC702-board onboard ADV7511 Chip and HDMI output connector.

The data flow of the system can be illustrated as follows:

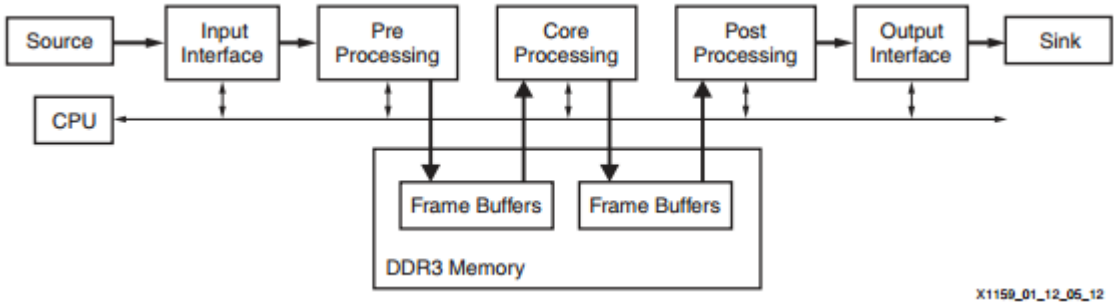


Figure 19: Data flow for video filter design

The video source is an internal test pattern generator or an external video source. The external video source is connected to the ADV7611 Chip using the FMC expansion card.

Pre Processing Stage

The preprocessing stage consist of five peripherals: (1) Video DMA, (2) Color Space Convertor (CSC), (3) Chroma Resampler (CRS), (4) Video Timing Controller (VTC) and (5) Test Pattern Generator (TPG).

A more detailed look at each IP core [7]:

AXI Video DMA

The AXI Video Direct Memory Access (AXI VDMA) core is a soft Xilinx IP core that provides high-bandwidth direct memory access between memory and AXI4-Stream type video target peripherals. The core provides efficient two dimensional DMA operations with independent asynchronous read and write channel operation. Initialization, status, interrupt and management registers are accessed through an AXI4-Lite slave interface.

In this system the VDMA is used to transfer incoming video data into DDR3 Memory frame buffers.

RGB to YCrCb Color-Space Converter

The Xilinx RGB to YCrCb Color-space Converter LogiCORE has built-in support for 5 formats and 3 range standards. The implementation is a simplified 3x3 constant coefficient matrix multiplier, which uses only 4 multipliers exploiting the inter-relations of RGB to YCrCb coefficients.

In this system the input video data is of the YCrCb format and is converted to the RGB format.

Chroma Resampler

The Chroma Resampler core converts between chroma formats of 4:4:4, 4:2:2, and 4:2:0. There are six conversions available for the three supported formats. Conversion is achieved using a FIR filter approach. Some conversions require filtering in only the horizontal dimension, vertical dimension, or both. Interpolation operations are implemented using a two-phase polyphase FIR filter. Decimation operations are implemented using a low-pass FIR filter to suppress chroma aliasing.

The Chroma Resampler converts the generated and converted RGB4:2:2 format to RGB4:4:4 format.

Video Timing Controller

The Xilinx Video Timing Controller LogiCORE IP is a general purpose video timing detector and generator, which automatically detects blanking and active data timing based on the input horizontal and vertical synchronization pulses. The Video Timing Controller can generate video timing signals and allows for adjustment of timing within a video design. The core is programmable through registers, provides a comprehensive set of interrupt controls, and supports multiple system configurations.

The Video Timing Controller is used to generate the necessary video timings for the internal test pattern generator.

Test Pattern Generator

The Xilinx Test Pattern Generator IP Core generates test patterns for Video System bring up, evaluation and debug. The core provides a wide variety of tests patterns enabling users to debug and asses video system color, quality, edge and motion performance and/or quality issues. The core can be inserted in an AXI4-Stream video interface that allows user selectable pass-through of system video signals or insertion of test patterns.

The test pattern generator provides a way to evaluate the system when no external video source is available. No additional effort is needed to generate a test pattern yourself.

Core Processing Stage

The core processing stage is the actual filter stage. In this example two filters are implemented: (1) Sobel filter and (2) Sepia filter. The filters are interchangeable by using partial reconfiguration.

Sobel filter

Sobel edge detection is a classical algorithm in the field of image and video processing for the extraction of object edges. Edge detection using Sobel operators works on the premise of computing an estimate of the first derivative of an image to extract edge information[8]. By computing the x and y direction derivatives of a specific pixel against a neighborhood of surrounding pixels, it is possible to extract the boundary between two distinct elements in an image. Due to the computational load of calculating derivatives using squaring and square root operators, fixed coefficient masks have been adopted as a suitable approximation in computing the derivative at a specific point.

The filter itself is generated using the Vivado High-Level Synthesis tool. The algorithm is fully written in C[9].

Sepia filter

The sepia filter applies a unique brown-tinted monochrome color to the input video stream. It adds the effect of older images to the video stream.

This filter is also generated using the Vivado High-Level Synthesis tool.

Video DMA

In order to transfer the video data to and from the DDR3 frame buffers a Video DMA core (in addition to the VDMA code used in the pre processing stage) is used. When no filter is applied the video data remains in the DDR3 frame buffer ready for the post-processing stage.

Figure 20: Filter engines: (1) Video Passthrough, (2) Sobel filter output and (3) Sepia filter output

Post Processing Stage

The Post Processing Stage consists of a CVCLogic display controller. This third-party IP core has a DMA engine embedded to read DDR frame buffers, a built-in color space conversion and chroma resampler unit. This is needed to convert the RGB4:4:4 format to the YCrCb4:2:2 format.

Partial Reconfiguration

The filter engines are reconfigurable. This means the filters cannot coexist at the same time, they are time-mutual exclusive. The Partial reconfiguration is done using the Xilinx Zynq Processor Configuration Access Port. The AXI-PCAP interface exists in the processing system and is connected to the general AXI interconnect network. It converts AXI packed data into PCAP data. Bitstream data is transferred to the PCAP interface using a built-in DMA engine. Throughput for non-secured bitstreams is about 400MB/s, for secured bitstreams the throughput is about 100MB/s.

Boot and Reconfiguration flow

When booting the system the internal Boot ROM checks external memory devices (SD Card, Flash, etc.) for a first stage boot loader (FSBL). The FSBL is transferred from the external memory device into on-chip memory (OCM) (phase 1). Next, the boot ROM exits and hands over CPU control to the FSBL. The FSBL configures the programmable logic with a bitstream (embedded in the FSBL) (phase 2). Next, the FSBL releases control and loads a baremetal application. This can be a user application or a second stage boot loader (such as U-Boot). The second stage boot loader can load a kernel image when using an embedded operating system.

On startup of the user application (whether it is a baremetal application or a OS-based application) partial bitstreams are loaded from external memory into DDR memory (phase 3). Reading the bitstream from DDR3 memory speeds up the reconfiguration process.

From now on, the application can load partial bitstreams through the PCAP interface when needed (phase 4). All phases are illustrated in next figure:

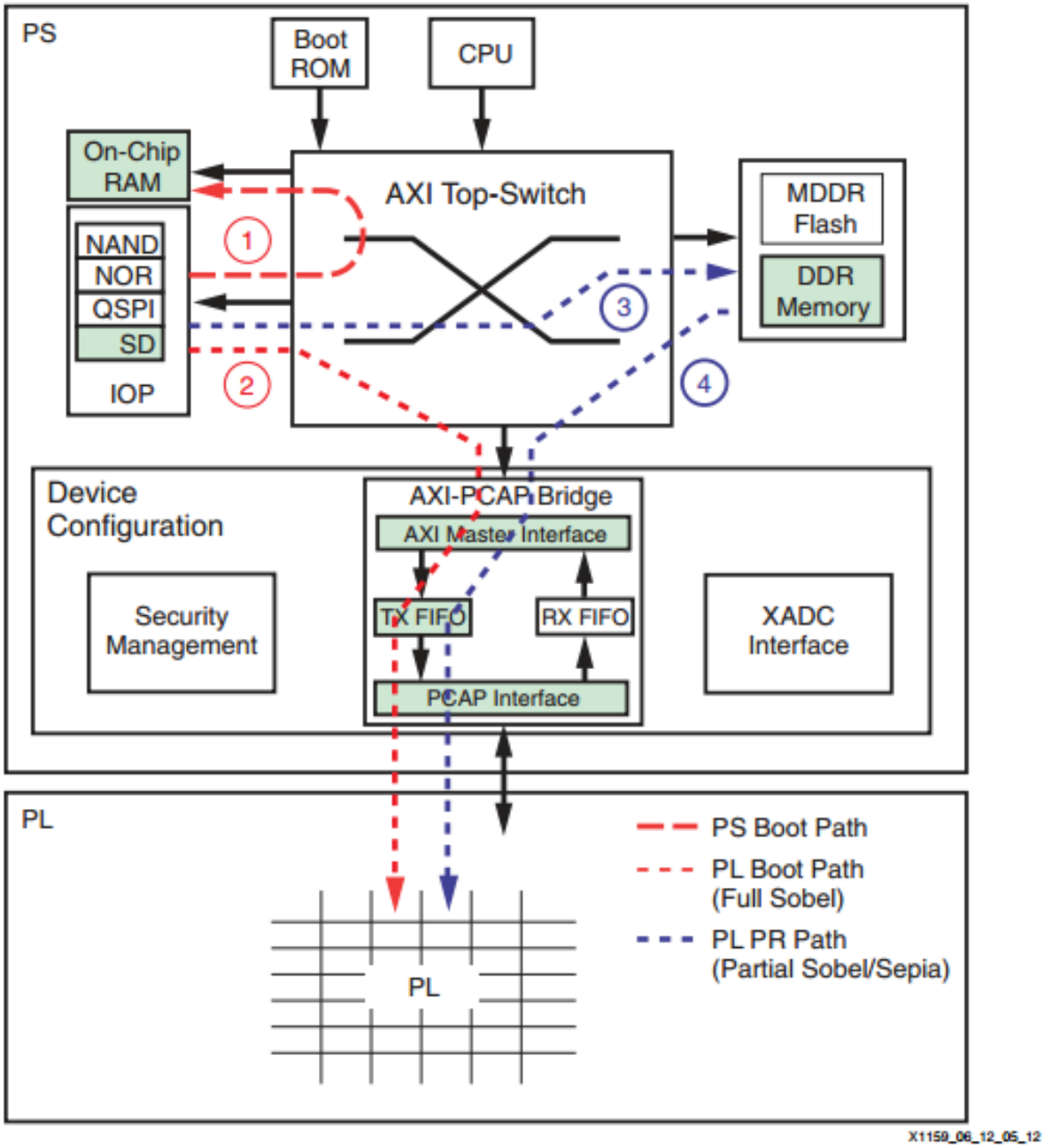


Figure 21: Device Configuration Flow (Boot and Reconfiguration)

ICAP Primitive

Besides all example projects presented by Xilinx where the ICAP peripheral is used in combination with a processor it is possible to instantiate the ICAP primitive in your HDL. In fact the ICAP peripheral is just a wrapper around the ICAP primitive. The

ICAP primitive is responsible of loading a partial bitstream but the method of storing the bitstream and the protocol to deliver it to the FPGA is up to the developer. The advantage is that the ICAP primitive is available for Spartan FPGA's too, not only the high-end devices.

ICAP interface

We examine the ICAP primitive for Virtex 6 devices (ICAP primitives for other devices are very similar, only bus width may vary). The primitive has four inputs: (1) clock, (2) ICAP Enable, (3) Read/Write control and (4) a data input bus. There are two outputs: (1) a busy/ready flag and (2) the output data bus. The interface is very similar to the SelectMap interface.

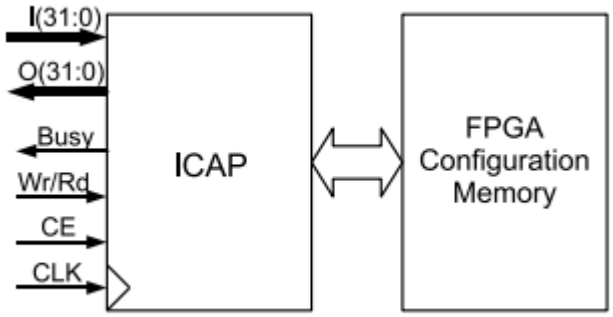


Figure 22: ICAP Primitive [10]

The ICAP primitive cannot be inferred (no Coregen support) and must be instantiated as following:

```
ICAP_VIRTEX6_inst : ICAP_VIRTEX6
generic map (ICAP_WIDTH => "X32“) -- Specifies I/O width
port map (
        BUSY => BUSY, -- 1-bit Busy(=0)/Ready(=1) output
        O => O, -- 32-bit Configuration data output bus
        CLK => CLK, -- 1-bit Clock Input
        CSB => CE, -- 1-bit Active-Low ICAP Enable
        I => I, -- 32-bit Configuration data input bus
        RDWRB => WE -- 1-bit Read(=1)/Write(=0) Select
);
```

A more detailed view on the interface:

PORT	Description	Additional Information
I(31:0)	Data to be loaded into configuration memory	ICAP_WIDTH attribute indicates a bus width of 8, 16, or 32 bits.
O(31:0)	Data read from configuration memory	ICAP_WIDTH applies to the output as well.

CLK	Clock input	Data must arrive synchronized with this clock. Output data is synchronized to this clock as well.
CSB	Active Low ICAP enable	Same as CS_B in the SelectMAP interface.
RDWRB	Read/write select	Low value indicates write.
BUSY	Active High indicates if ICAP is currently being read	Always Low when writing to ICAP

Not much information is provided to develop systems based on the ICAP primitive but Application Note 887 provides a good starting point [11].

Application Note 887 describes a partial reconfiguration controller (PRC) that can be included in any PR design to process partial bitstreams for data integrity. This is done by calculating the CRC of the partial bitstreams before they enter the ICAP primitive.

In standard configuration CRC checking is done when configuring the device. For partial bitstreams this cannot be done since the ICAP primitive lacks a buffering mechanism.

The protocol used to communicate with the ICAP primitive is identical to the SelectMap protocol.

The ICAP primitive output port provides the status during the reconfiguration. The bitstream should be provided on the data input pins without wait states. The busy/ready pin and the read/write pin can be left unconnected if configuration readback is not needed. The ICAP enable pin is used to enable the ICAP interface. If the interface is disabled all data will be ignored. All activity on the ICAP interface is synchronous to the clock.

Additional remark is that the ICAP primitive is not considered as a synchronous element. Paths to and from the ICAP are not covered by PERIOD constraints, ICAP cannot become synchronous via the use of the TPSYNC constraint.

Simulating Partial Reconfigurable Designs

Introduction

According to Xilinx UG702 [12] p. 116, all standard simulation, timing analysis, and verification techniques are supported for PR designs. Partial reconfiguration itself cannot be simulated. Rousseau et al [13] pointed the difficulties concerning the inability to simulate PR:

[…] Developers using PR must complete implementation and integration of the system before being able to test it. This forces developers to jump from development to integration without having processed to complete validation. The lack of validation is a major issue which can make development with PR infeasible or inefficient and difficult for professional uses where a design must be strictly validated at each step

of the work. […]

In the past, several research efforts are targeting the simulation issue for PR designs. Examples are [14], [15], [16], [17], [18]. However recently Gong. et al [18] provided a way to simulate PR designs on Register Transfer Level (RTL) in a cycle-accurate way. Gong et al [18] developed a library named ReSim which provides a simulation-only layer on top of a PR design.

ReSim

The ReSim library is built upon the SystemVerilog verification language on the Open Verification Methodology (OVM) library. Therefore it is easily integratable into industry standard EDA tools such as ModelSim. Since the reconfiguration process involves Application Layer, reconfiguration Layer and Physical Layer it is challenging to simulate the PR process. To get rid of the physical dependencies the configuration port and configuration memory are emulated by an ICAP artifact and an extended portal. When the user application needs reconfiguration a Simulation-only bitstream (SimB) is transferred from storage (extended portal) to configuration port (ICAP artifact).The extended portal has the ability to inject errors into the SimB in order to simulate functional behavior. The SimB is a simulation-only bitstream with consists of a header, configuration part and tail. This is similar to a real bitstream but is significantly smaller in size. The configuration part of the bitstream is usually filled with physical dependent information. In the case of the SimB it is filled with dummy-data.

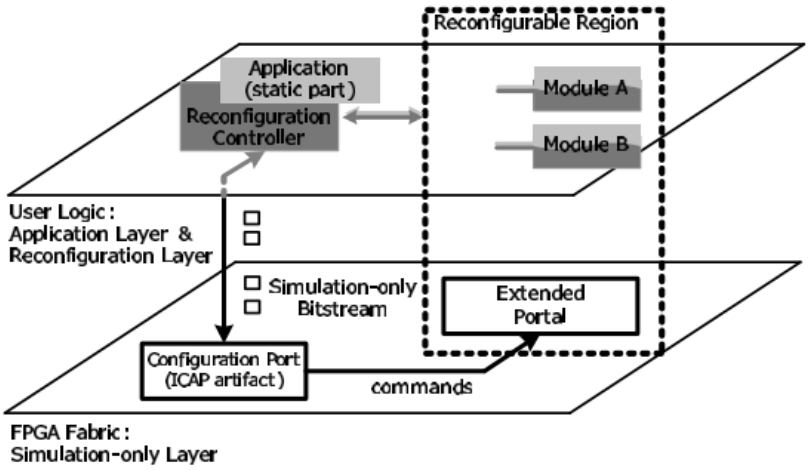


Figure 23. ReSim Simulation-only layer

Example PCI Express design

The PCI Express design is borrowed from Xilinx xapp883 [19]. Partial Reconfiguration is the ideal solution for meeting the 100ms startup requirements for PCI Express devices. For large designs the timing requirement can be difficult to meet, therefore a two-stage bring-up is implemented using partial reconfiguration. In

general the initial configuration consists only of the needed parts, such as PCI Express IP core, a switcher for managing ICAP traffic and user application traffic. The user application itself is initially not configured, instead a blank bitstream is used. After the communication link is established, the full user application is configured through the PCI Express interface. The overall module architecture is illustrated in Figure 24. PCI Express Module Architecture [7]

The BAR0 address range (1kB) is used for addressing the user applications. BAR2 address range (2kB) is used for the PR Loader part of the system.

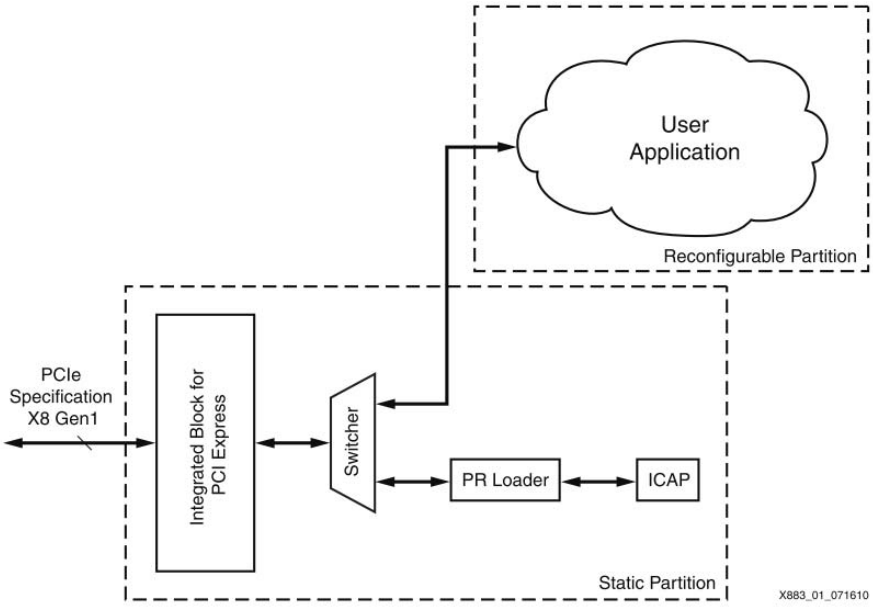


Figure 24. PCI Express Module Architecture [7]

Conclusion

ReSim provides a powerfull way of functional verification of PR design. However, as for all RTL functional simulation, it does not guarantee a correct functioning in hardware. The use of a SimB can be a source of different behavior between the simulated design and the implemented design.

Safety-Certification

Certification for FPGA design brings additional difficulties since FPGA's are (re-) programmable, they have a configuration memory and use certain mechanisms to configure itself. Typically techniques such as modular design and inter-module communication by external IO ports help in the certification of complex FPGA designs. As independence between modules is guaranteed, failure contagion is avoided. Typically third-party IP's and tools need be certified to in order to use them for a certified FPGA design.

In order to use partial reconfiguration in critical applications different problems arise:

- How to certify the proprietary partial reconfiguration toolflow?

- Validation of the partial bitstreams
- ICAP verification
- Transient state during reconfiguration
- Different functions for a single module
- High complexity of the design

According to Rousseau et al[13]. the minimal flow for safety certification is generate partial bitstreams from complete bitstreams generated with the static design flow. The only additional operations needed are binary file parsing tools, packet ordering and file writing. The reconfiguration itself is done using an external configuration interface. The rest of the component is stopped when reconfiguring.

Bibliography

[1] PCI SIG, *PCI Express Base Specification*. 2012.

[2] D. Koch, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications (Lecture Notes in Electrical Engineering)*. Springer, 2012, p. 317.

[3] A. K. Kurbjun and C. Ribbing, “Xilinx XAPP878: MMCM Dynamic Reconfiguration,” vol. 878, pp. 1–13, 2010.

[4] Altera, “Dynamic Reconfiguration in Stratix IV,” *Stratix V handbook*, vol. 2, no. September. 2012.

[5] Xilinx, “DS817: LogiCORE IP AXI HWICAP (v2.03.a),” vol. DS817. 2012.

[6] C. Kohn, “Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices,” *Application Note: Zynq-7000 All Programmable SoC*, vol. XAPP1159, no. UG1159. pp. 1–19, 2013.

[7] Xilinx, “Video IP and CODEC IP,” 2013. [Online]. Available: http://www.xilinx.com/esp/video/refdes_listing.htm#video. [Accessed: 09-Dec-2013].

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson, 2008, p. 976.

[9] F. M. Vallina, C. Kohn, and P. Joshi, “Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool,” vol. XAPP890, pp. 1–16, 2012.

[10] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, “Run-Time Partial Reconfiguration Speed Investigation and Architectural Design Space Exploration,” in *Proceedings of the IEEE International Workshop on Field Programmable Logic and Applications (FPL)*, 2009.

[11] Xilinx, “PRC / EPRC : Data Integrity and Security,” vol. XAPP887. pp. 1–17, 2012.

[12] Xilinx, “Partial Reconfiguration User Guide,” vol. UG702. 2010.

[13] B. Rousseau, P. Manet, T. Delavallée, I. Loiselle, and J.-D. Legat, “Dynamically Reconfigurable Architectures for Software-Defined Radio in Professional Electronic Applications,” in in *Design Technology for Heterogeneous Embedded Systems*, Springer, 2011, pp. 437–457.

[14] A. Schallenberg, W. Nebel, A. Herrholz, P. A. Hartmann, and F. Oppenheimer, “OSSS+R: A framework for application level modelling and synthesis of reconfigurable systems,” in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, pp. 970–975.

[15] A. Raabe and A. Felke, “A SystemC Language Extension for High-Level Reconfiguration Modelling,” *Forum on Specification and Design Languages*, pp. 55–60, 2008.

[16] I. Robertson, J. Irvine, P. Lysaght, G. Street, and D. Robinson, “Timing Verification of Dynamically Reconfigurable Logic for the Xilinx Virtex FPGA Series,” 2002.

[17] J. Stockwood and P. Lysaght, “A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays,” 1995.

[18] L. Gong and O. Diessel, “ReSim: A reusable library for RTL simulation of dynamic partial reconfiguration,” in *2011 International Conference on Field-Programmable Technology*, 2011, pp. 1–8.

[19] M. Kellermann, “Fast Configuration of PCI Express,” vol. XAPP883, pp. 1–45, 2010.